

The Scalable Modeling System (SMS): A High-Level Directive- Based Alternative to MPI

Tom Henderson

Mark Govett

Leslie Hart

Jacques Middlecoff

Dan Schaffer

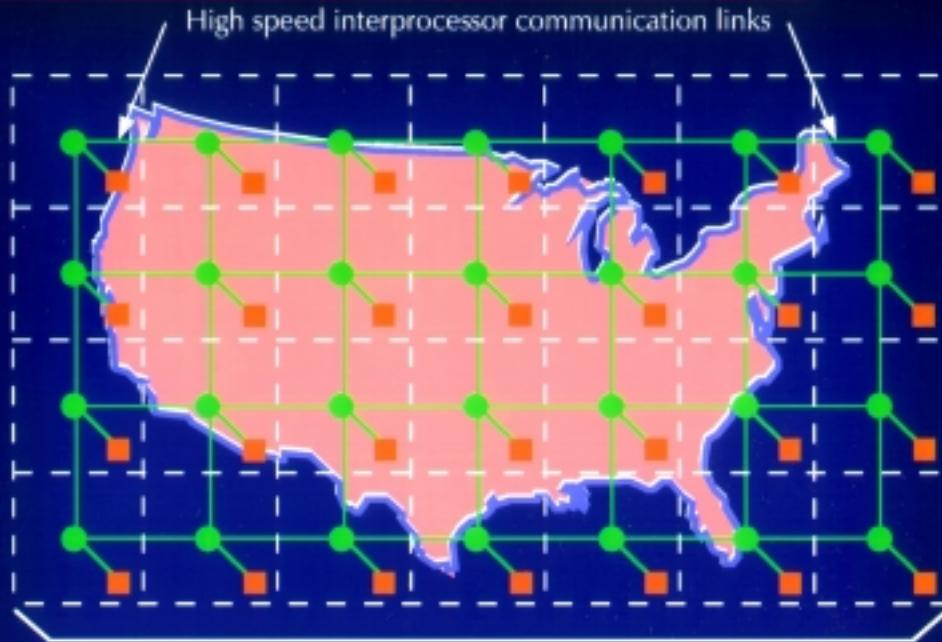
NOAA Forecast Systems Laboratory

May 2000

Outline

- Overview/SMS Structure
- What SMS Does
- Short Code Example
- Results and Preliminary Performance
- Summary
- On-going Development

MPP Weather Prediction with SMS

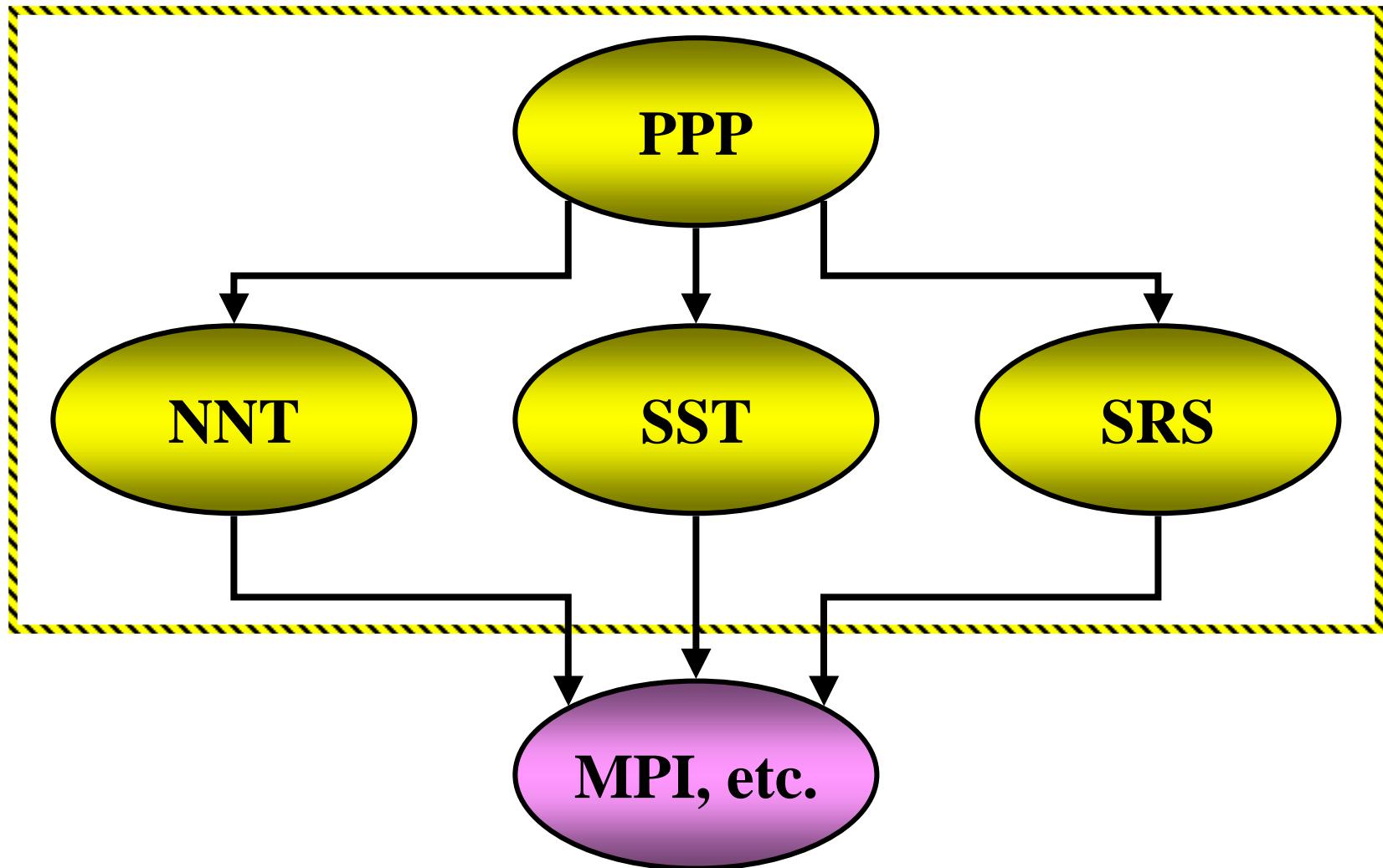


FORECAST
SYSTEMS
LABORATORY
BOULDER, COLORADO

SMS Goals

- High performance
- Ease of use
- Portability
 - Distributed and/or shared memory
- Interoperability
 - MPI, OpenMP, etc.
- Focus on numerical weather prediction (NWP) models
 - Finite difference approximation (FDA)
 - Gauss-Legendre transforms (spectral)
 - Fortran 77 (Fortran 90 under development)

SMS Components



SMS Components

- PPP: Parallel Pre-Processor
 - User inserts comment-based directives
CSMS\$EXCHANGE(u , v , t)
 - PPP automatically generates parallel code
 - Most users only need this component

SMS Low-level Components

- Not needed by most users
- Available via subroutine calls
- Callable from F90 code
- NNT: Nearest Neighbor Tool
 - Low-level support for finite-difference models
- SST: Scalable Spectral Tool
 - Low-level support for spectral transform models
- SRS: Scalable Run-time System
 - Low-level support for fast parallel I/O

A Quick Peek Ahead for the Eager

```
integer IM, i
parameter(IM = 15)
CSMS$DECLARE_DECOMP(my_dh, <(IM/3) + 4>)

CSMS$DISTRIBUTE(my_dh, <IM>) BEGIN
    real x(IM), y(IM), xsum
CSMS$DISTRIBUTE END

C Begin executable code
CSMS$CREATE_DECOMP (my_dh, <IM>, <2>

    open (10, file = 'x_in.dat', form='unformatted')
    read (10) x
    close (10)

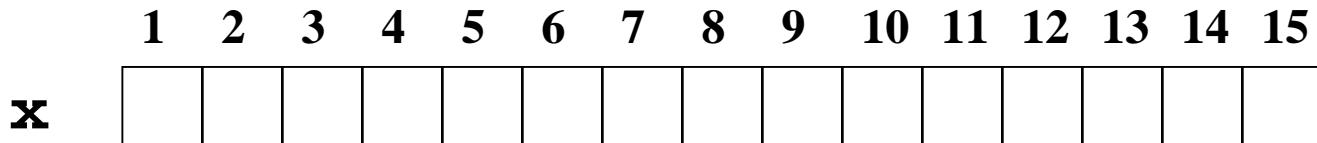
CSMS$PARALLEL(my_dh, <i>) BEGIN
    do 100 i = 3, 13
        y(i) = x(i) - x(i-1) - x(i+1) - x(i-2) - x(i+2)
100    continue
CSMS$EXCHANGE(y)
    do 200 i = 3, 13
        x(i) = y(i) + y(i-1) + y(i+1) + y(i-2) + y(i+2)
200    continue
    xsum = 0.0
    do 300 i = 1, 15
        xsum = xsum + x(i)
300    continue
CSMS$REDUCE(xsum, SUM)
CSMS$PARALLEL END
    print *, 'xsum = ', xsum
```

What Does SMS Do?

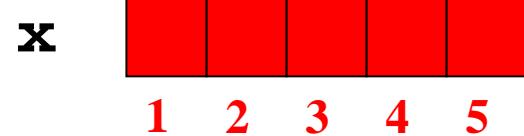
- Data decomposition
- Memory layout
- I/O
- Loop transformation
- Index translation
- Inter-process communication
- Static load balancing
- Performance optimization
- Debugging support

Simple 1D Decomposition

real x(15)

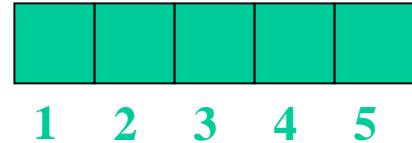


real x(5)



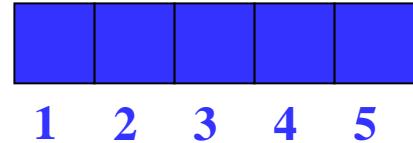
P1

real x(5)



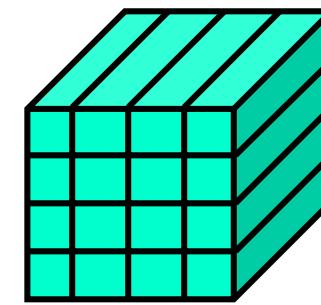
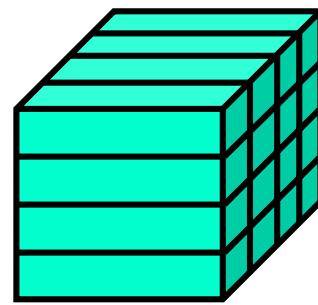
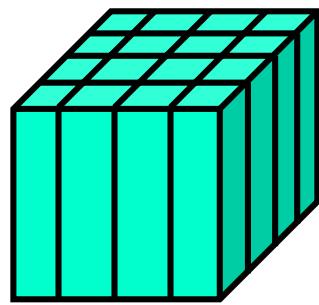
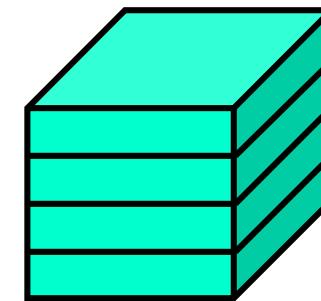
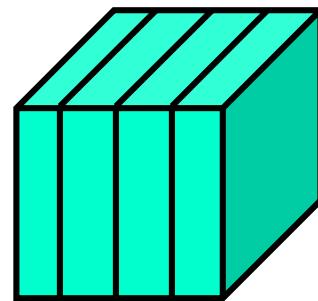
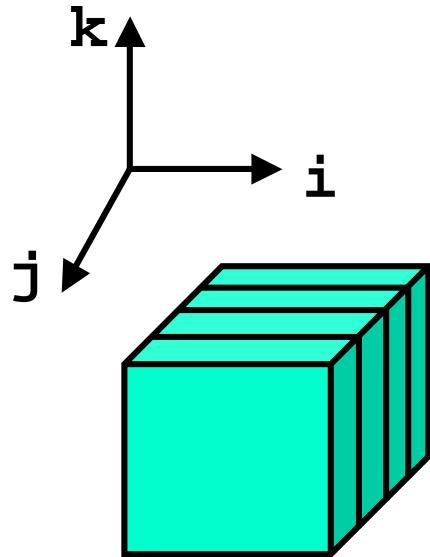
P2

real x(5)



P3

Common Decompositions



SMS Directives:

`CSMS$DECLARE_DECOMP , CSMS$CREATE_DECOMP`

What Does SMS Do?

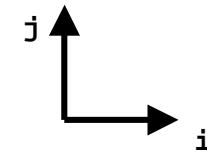
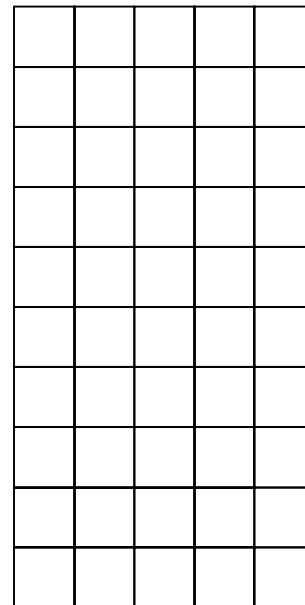
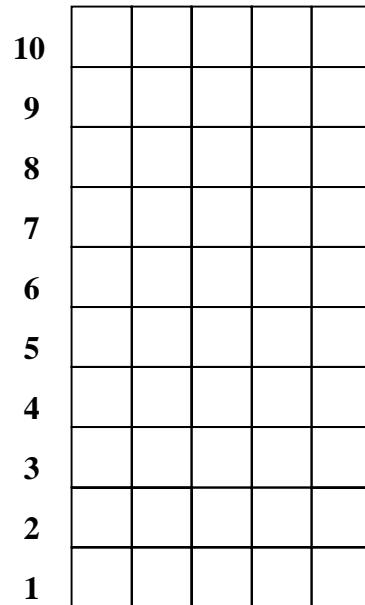
- Data decomposition
- **Memory layout**
- I/O
- Loop transformation
- Index translation
- Inter-process communication
- Static load balancing
- Performance optimization
- Debugging support

Memory Layout

“Local”
declarations:

integer x(5,10)

integer x(5,10)



“Local” indices: 1 2 3 4 5

“Global” indices: 1 2 3 4 5 6 7 8 9 10

PROCESS:

P1

P2

SMS Directive: **CSMS\$DISTRIBUTE**

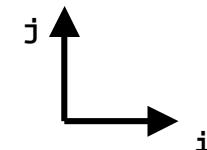
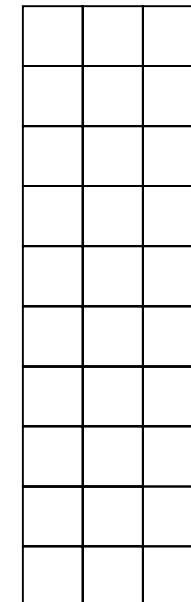
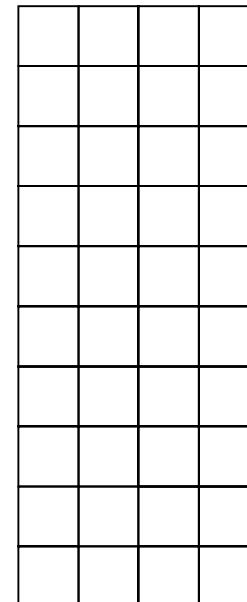
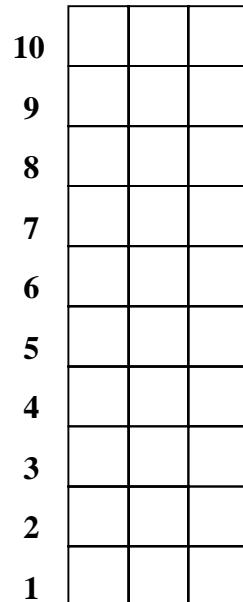
Memory Layout

“Local”
declarations:

integer x(3,10)

integer x(3,10)

integer x(4,10)



“Local” indices:

1 2 3

1 2 3 4

1 2 3

“Global” indices:

1 2 3

4 5 6 7

8 9 10

PROCESS:

P1

P2

P3

SMS Directive: **CSMS\$DISTRIBUTE**

What Does SMS Do?

- Data decomposition
- Memory layout
- I/O
- Loop transformation
- Index translation
- Inter-process communication
- Static load balancing
- Performance optimization
- Debugging support

Simple Non-Decomposed I/O

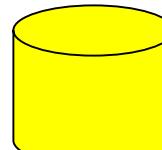
real g

read (10) g

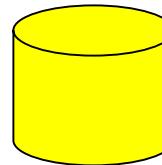
g

P1

write (10) g

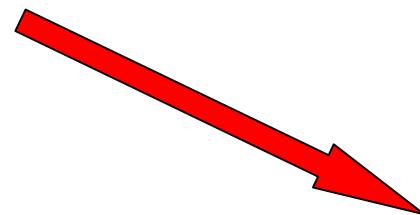
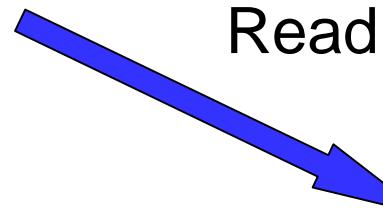
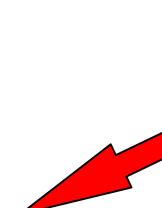


P2



P3

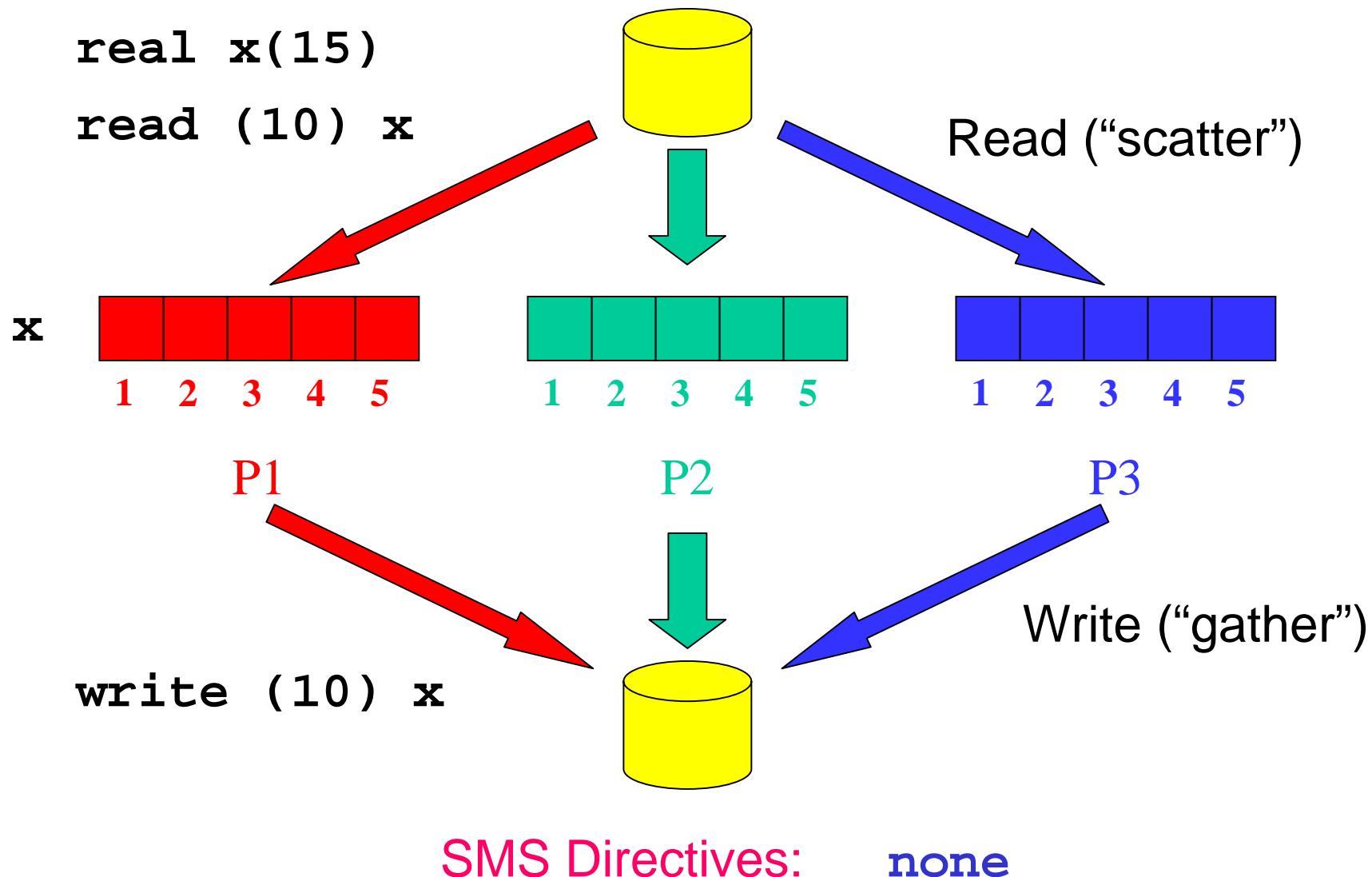
Read (“broadcast”)



Write (“root”)

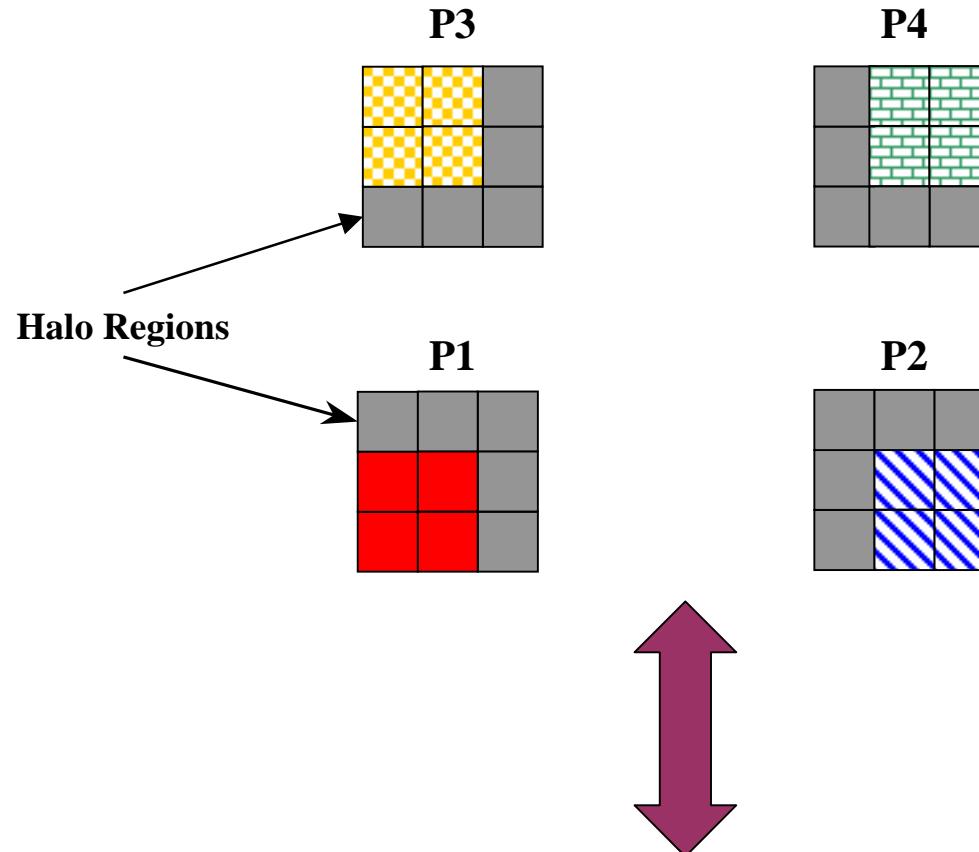
SMS Directives: none

Simple Decomposed I/O



I/O of Decomposed Arrays

PARALLEL MEMORY LAYOUT



SERIAL FILE DATA LAYOUT



SMS Unformatted I/O

- Default file format is native Fortran binary
- Other file formats supported including portable MPI-IO External
 - Select file format at **run time** via environment var.
- Serial data ordering
- No directives required
- OK to mix non-decomposed and decomposed variables
- SMS programs can read any file written by an SMS program in MPI format regardless of the machine or number of processes that wrote it

Printing

- What does parallel print mean?

```
print *, 'hello'  
>> hello  
>> hellohellohello  
>> hhelheellollloo  
>> hello  
    hello  
    hello
```

- Three print modes
 - Default mode mimics serial code
 - Other modes selected via directive

SMS Directive: **CSMS\$PRINT_MODE**

What Does SMS Do?

- Data decomposition
- Memory layout
- I/O
- **Loop transformation**
- Index translation
- Inter-process communication
- Static load balancing
- Performance optimization
- Debugging support

Loop Transformation

```
C original serial loop
    do i=1,15

C parallel loops over the sub-domain of each process
C P1 -->
    do i=1,4
C P2 -->
    do i=1,4
C P3 -->
    do i=1,4
C P4 -->
    do i=1,3
```

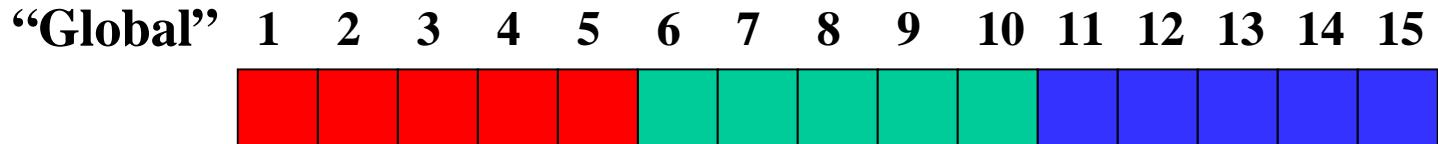
SMS Directive: **CSMS\$PARALLEL**

What Does SMS Do?

- Data decomposition
- Memory layout
- I/O
- Loop transformation
- **Index translation**
- Inter-process communication
- Static load balancing
- Performance optimization
- Debugging support

Global and Local Indices

`real x(15)`



P1

P2

P3

SMS Directives:

`CSMS$TO_LOCAL`

`CSMS$TO_GLOBAL`

`CSMS$GLOBAL_INDEX`

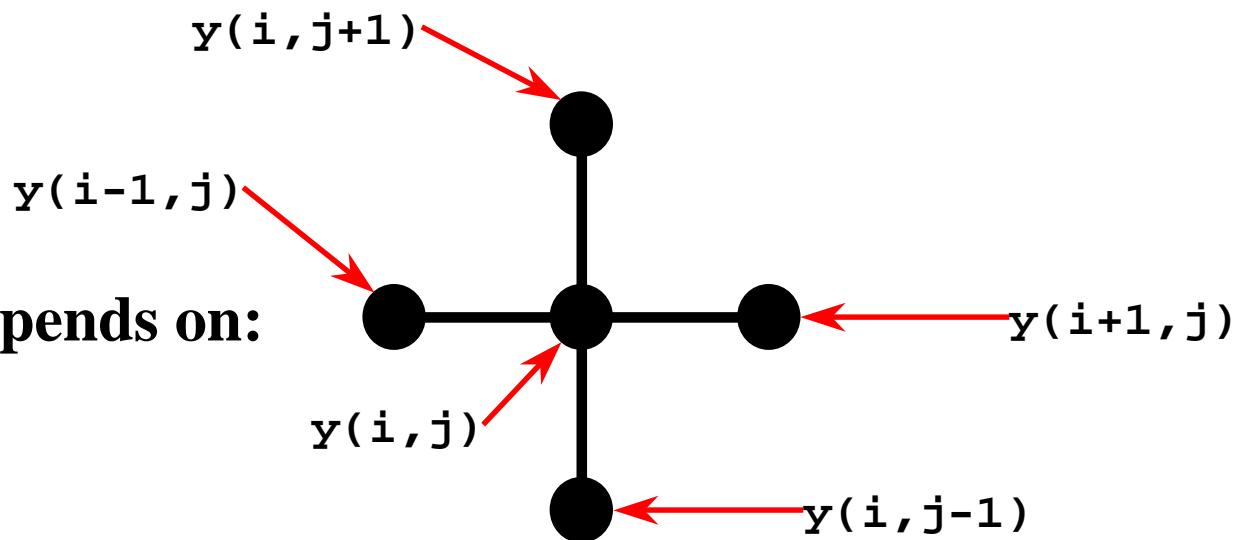
What Does SMS Do?

- Data decomposition
- Memory layout
- I/O
- Loop transformation
- Index translation
- **Inter-process communication**
- Static load balancing
- Performance optimization
- Debugging support

Finite-difference Dependencies

$$x(i,j) = y(i,j) + y(i+1,j) + y(i-1,j) + y(i,j-1) + y(i,j+1)$$

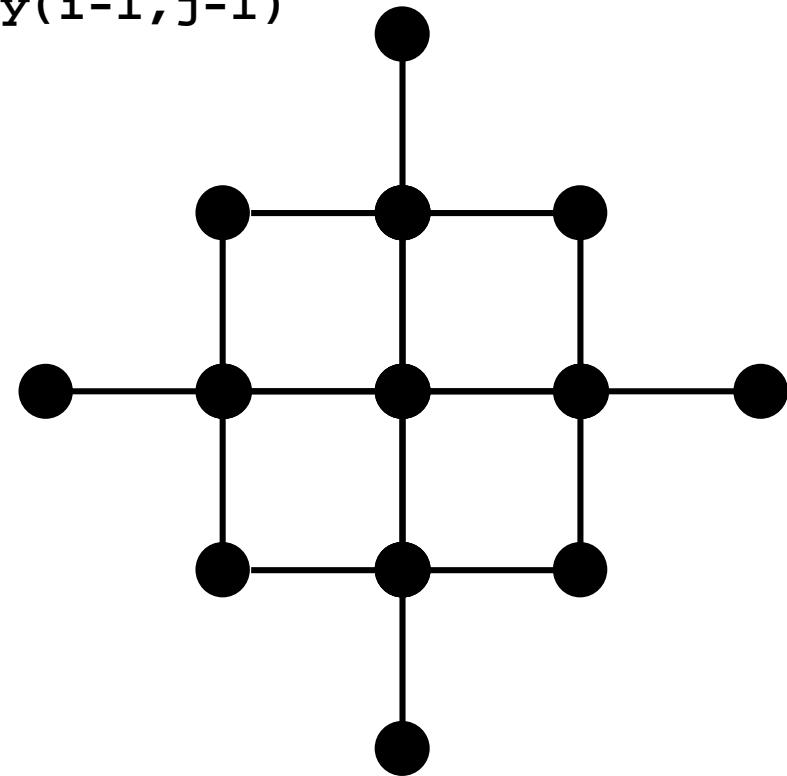
“Stencil”: $x(i,j)$ depends on:



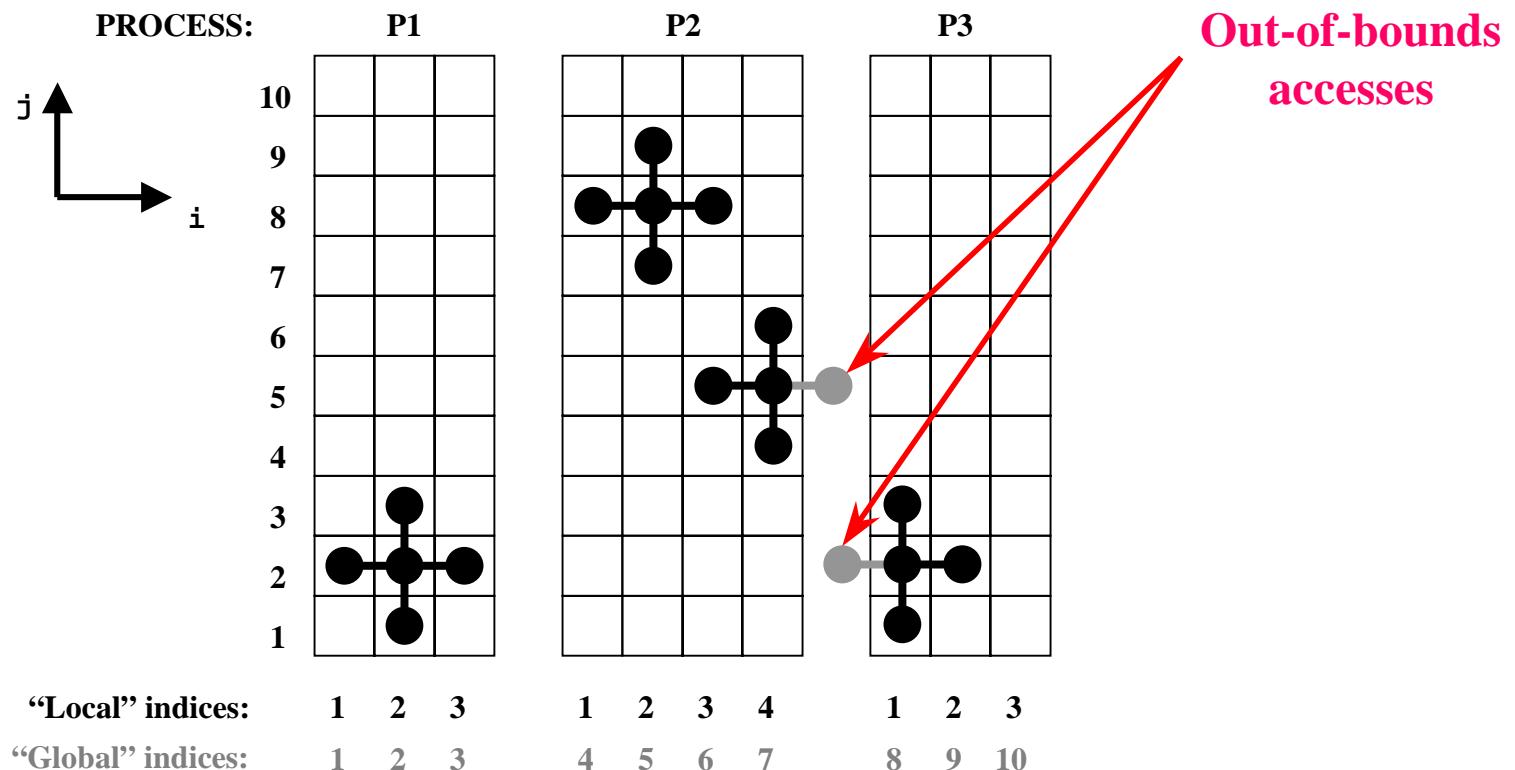
Finite-difference Dependencies

```
x(i,j) = y(i,j) + y(i+1,j) + y(i+2,j) + y(i,j+1) + y(i,j+2)  
       + y(i-1,j) + y(i-2,j) + y(i,j-1) + y(i,j-2)  
       + y(i+1,j+1) + y(i+1,j-1)  
       + y(i-1,j+1) + y(i-1,j-1)
```

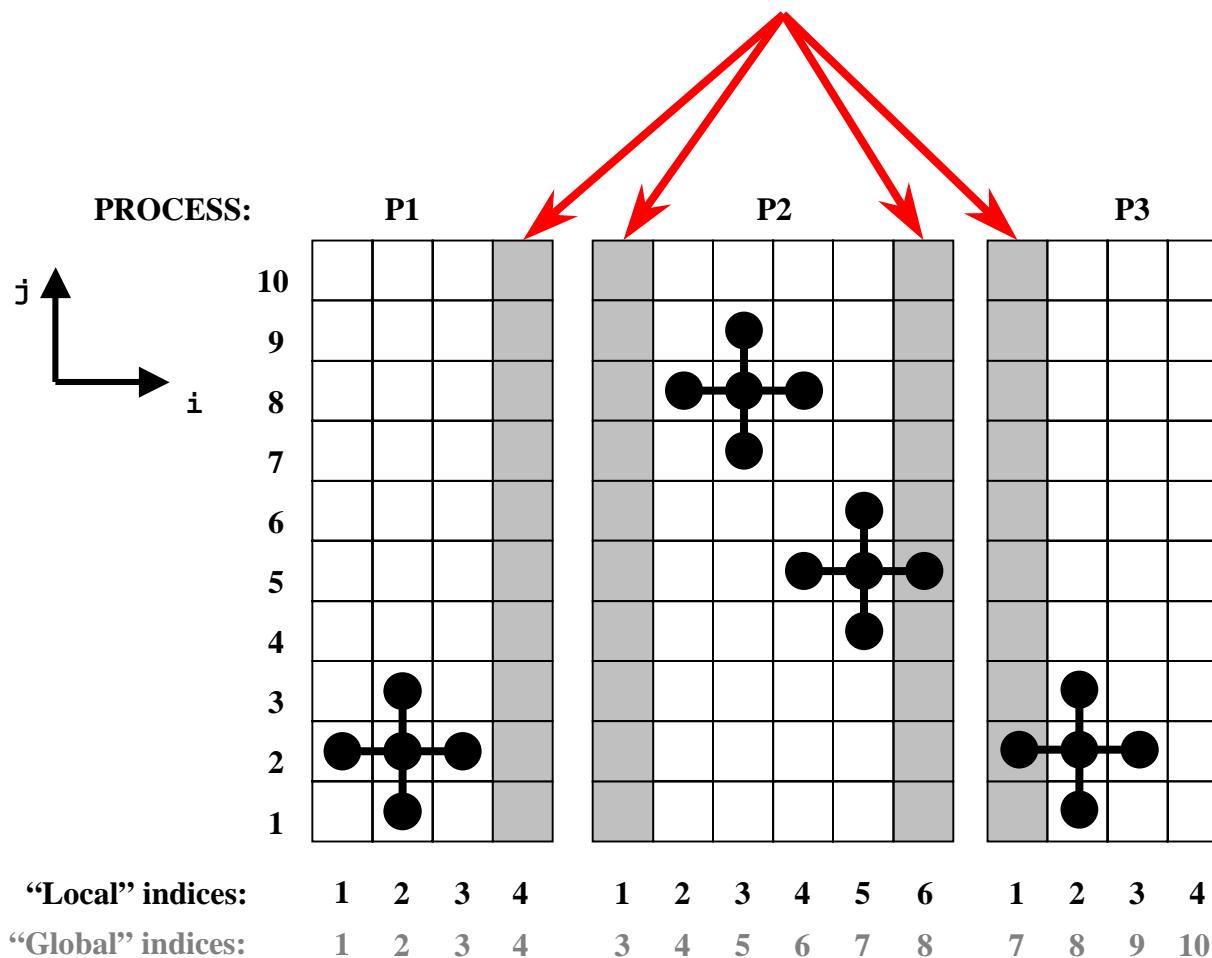
“Stencil”: $x(i,j)$ depends on:



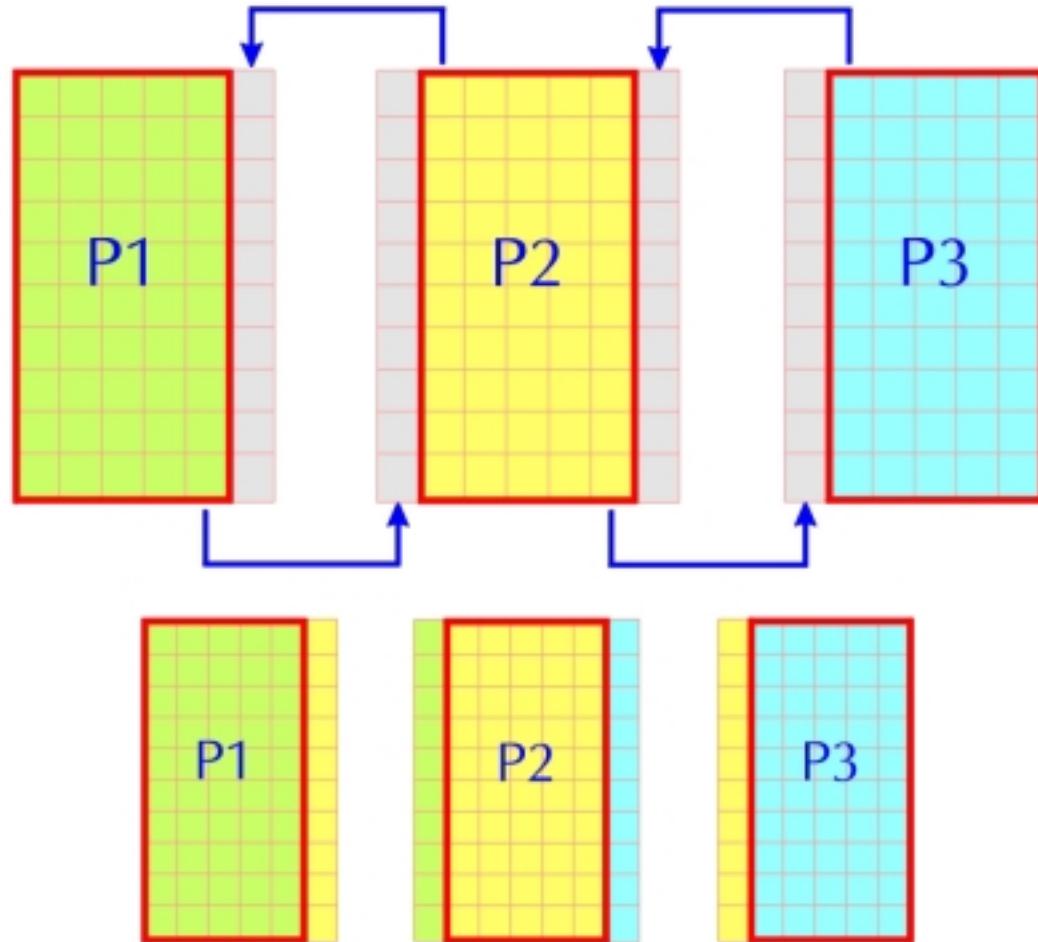
Unresolved Dependencies



"Halo" Regions

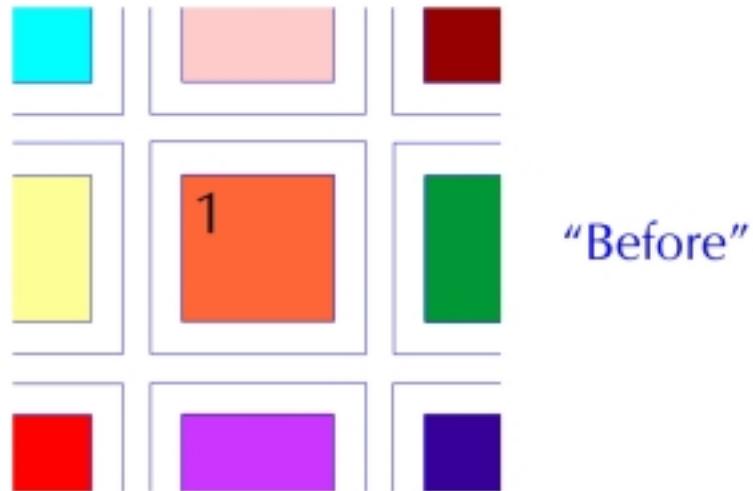


“Exchange”

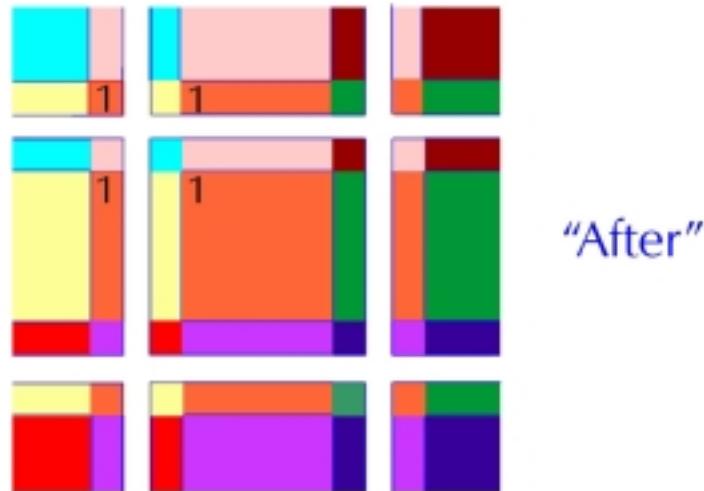


SMS Directive: **CSMS\$EXCHANGE**

2D Exchange



2D Exchange



A Simple Hand-coded Exchange

```
SUBROUTINE EXCH(ARR,LL,IHALO,JHALO)
INTEGER ISTAT(MPI_STATUS_SIZE),STATUS_ARRAY(MPI_STATUS_SIZE,4)
INTEGER IHANDLE(4)
REAL ARR(IDIM1:IDIM2,JDIM1:JDIM2,LL)
REAL,ALLOCATABLE,DIMENSION(:,:,:,:) :: BUF0,BUF1,BUF2,BUF3
ITYPE=MPI_REAL
C-----
C      RECEIVE FROM NORTH
C-----
IF(MY_NEB(1).GE.0)THEN
    NEBPE=MY_NEB(1)
    IBEG=MYIS-IHALO
    IEND=MYIE+IHALO
    ISIZE=(IEND-IBEG+1)*JHALO*LL
    ALLOCATE(BUF0(IBEG:IEND,0:JHALO-1,LL),STAT=I)
    CALL MPI_IRecv(BUF0,ISIZE,ITYPE,NEBPE,NEBPE
1,           MPI_COMM_WORLD,IHANDLE(1),IRECV)
    ENDIF
C-----
C      RECEIVE FROM SOUTH
C-----
IF(MY_NEB(3).GE.0)THEN
    NEBPE=MY_NEB(3)
    IBEG=MYIS-IHALO
    IEND=MYIE+IHALO
    ISIZE=(IEND-IBEG+1)*JHALO*LL
    ALLOCATE(BUF1(IBEG:IEND,0:JHALO-1,LL),STAT=I)
    CALL MPI_IRecv(BUF1,ISIZE,ITYPE,NEBPE,NEBPE
1,           MPI_COMM_WORLD,IHANDLE(2),IRECV)
    ENDIF
```

A Simple Hand-coded Exchange

```
C-----  
C      SEND TO NORTH  
C-----  
IF(MY_NEB(1).GE.0)THEN  
  NEBPE=MY_NEB( 3 )  
  NEBPE=MY_NEB(1)  
  IBEG=MYIS-IHALO  
  IEND=MYIE+IHALO  
  ALLOCATE(BUF2( IBEG:IEND , 0:JHALO-1 , LL ), STAT=I )  
  KNT=( IEND-IBEG+1)*JHALO*LL  
  DO K=1,LL  
    DO J=0,JHALO-1  
      DO I=IBEG,IEND  
        BUF2(I,J,K)=ARR(I,MYJE-J,K)  
      ENDDO  
    ENDDO  
  ENDDO  
  CALL MPI_ISEND(BUF2,KNT,ITYPE,NEBPE,MYPE  
1,           MPI_COMM_WORLD,IHANDLE(3),ISEND)  
ENDIF
```

A Simple Hand-coded Exchange

```
C-----  
C      SEND TO SOUTH  
C-----  
  
IF(MY_NEB(3).GE.0)THEN  
    NEBPE=MY_NEB(3)  
    IBEG=MYIS-IHALO  
    IEND=MYIE+IHALO  
    ALLOCATE(BUF3(IBEG:IEND,0:JHALO-1,LL),STAT=I)  
    KNT=(IEND-IBEG+1)*JHALO*LL  
    DO K=1,LL  
        DO J=0,JHALO-1  
            DO I=IBEG,IEND  
                BUF3(I,J,K)=ARR(I,MYJS+J,K)  
            ENDDO  
        ENDDO  
    ENDDO  
    CALL MPI_ISEND(BUF3,KNT,ITYPE,NEBPE,MYPE  
1,                      MPI_COMM_WORLD,IHANDLE(4),ISEND)  
ENDIF
```

A Simple Hand-coded Exchange

```
C-----
C      STORE RESULTS FROM SOUTH
IF(MY_NEB(3).GE.0)THEN
    IBEG=MYIS-IHALO
    IEND=MYIE+IHALO
    CALL MPI_WAIT(IHANDLE(2),ISTAT,IERR)
    DO K=1,LL
        DO J=0,JHALO-1
            DO I=IBEG,IEND
                ARR(I,MYJS-J-1,K)=BUF1(I,J,K)
            ENDDO
        ENDDO
    ENDDO
    DEALLOCATE(BUF1,STAT=IER)
ENDIF
C-----
C      STORE FROM NORTH
IF(MY_NEB(1).GE.0)THEN
    IBEG=MYIS-IHALO
    IEND=MYIE+IHALO
    CALL MPI_WAIT(IHANDLE(1),ISTAT,IERR)
    DO K=1,LL
        DO J=0,JHALO-1
            DO I=IBEG,IEND
                ARR(I,MYJE+J+1,K)=BUF0(I,J,K)
            ENDDO
        ENDDO
    ENDDO
    DEALLOCATE(BUF0,STAT=IER)
ENDIF
```

A Simple Hand-coded Exchange

```
IF(MY_NEB(1).GE.0)THEN
    CALL MPI_WAIT(IHANDLE(3),ISTAT,IERR)
    DEALLOCATE(BUF2,STAT=IER)
ENDIF
IF(MY_NEB(3).GE.0)THEN
    CALL MPI_WAIT(IHANDLE(4),ISTAT,IERR)
    DEALLOCATE(BUF3,STAT=IER)
ENDIF
C-----
C      RECEIVE FROM WEST
IF(MY_NEB(4).GE.0)THEN
    NEBPE=MY_NEB(4)
    IBEG=MYIS-IHALO
    IEND=IBEG+IHALO-1
    ISIZE=(IEND-IBEG+1)*(MYJE+JHALO-MYJS+JHALO+1)*LL
    ALLOCATE(BUF0(IBEG:IEND,MYJS-JHALO:MYJE+JHALO,LL),STAT=I)
    CALL MPI_RECV(BUF0,ISIZE,ITYPE,NEBPE,NEBPE
1,           MPI_COMM_WORLD,IHANDLE(1),IRECV)
ENDIF
C-----
C      RECEIVE FROM EAST
IF(MY_NEB(2).GE.0)THEN
    NEBPE=MY_NEB(2)
    IBEG=MYIE+1
    IEND=MYIE+IHALO
    ISIZE=(IEND-IBEG+1)*(MYJE+JHALO-MYJS+JHALO+1)*LL
    ALLOCATE(BUF1(IBEG:IEND,MYJS-JHALO:MYJE+JHALO,LL),STAT=I)
    CALL MPI_RECV(BUF1,ISIZE,ITYPE,NEBPE,NEBPE
1,           MPI_COMM_WORLD,IHANDLE(2),IRECV)
ENDIF
```

A Simple Hand-coded Exchange

```
C-----  
C      SEND TO EAST  
C-----  
  
IF(MY_NEB(2).GE.0)THEN  
  NEBPE=MY_NEB(2)  
  IBEG=MYIE-IHALO+1  
  IEND=MYIE  
  KNT=(IEND-IBEG+1)*(MYJE+JHALO-MYJS+JHALO+1)*LL  
  ALLOCATE(BUF2(IBEG:IEND,MYJS-JHALO:MYJE+JHALO,LL),STAT=I)  
  DO K=1,LL  
    DO J=MYJS-JHALO,MYJE+JHALO  
      DO I=IBEG,IEND  
        BUF2(I,J,K)=ARR(I,J,K)  
      ENDDO  
    ENDDO  
  ENDDO  
  CALL MPI_ISEND(BUF2,KNT,ITYPE,NEBPE,MYPE  
1,           MPI_COMM_WORLD,IHANDLE(3),ISEND)  
ENDIF
```

A Simple Hand-coded Exchange

```
C-----  
C      SEND TO WEST  
C-----  
IF(MY_NEB(4).GE.0)THEN  
  NEBPE=MY_NEB(4)  
  IBEG=MYIS  
  IEND=MYIS+JHALO-1  
  KNT=(IEND-IBEG+1)*(MYJE+JHALO-MYJS+JHALO+1)*LL  
  ALLOCATE(BUF3(IBEG:IEND,MYJS-JHALO:MYJE+JHALO,LL),STAT=I)  
  DO K=1,LL  
    DO J=MYJS-JHALO,MYJE+JHALO  
      DO I=IBEG,IEND  
        BUF3(I,J,K)=ARR(I,J,K)  
      ENDDO  
    ENDDO  
  ENDDO  
  CALL MPI_ISEND(BUF3,KNT,ITYPE,NEBPE,MYPE  
1,           MPI_COMM_WORLD,IHANDLE(4),ISEND)  
ENDIF
```

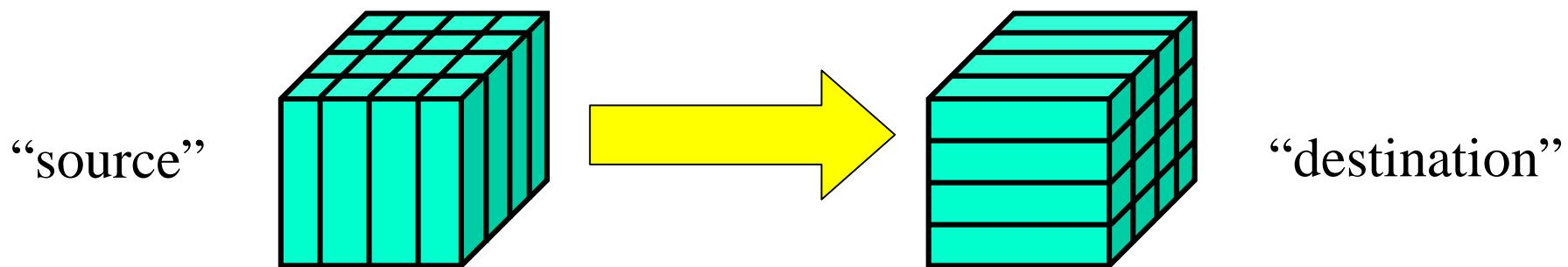
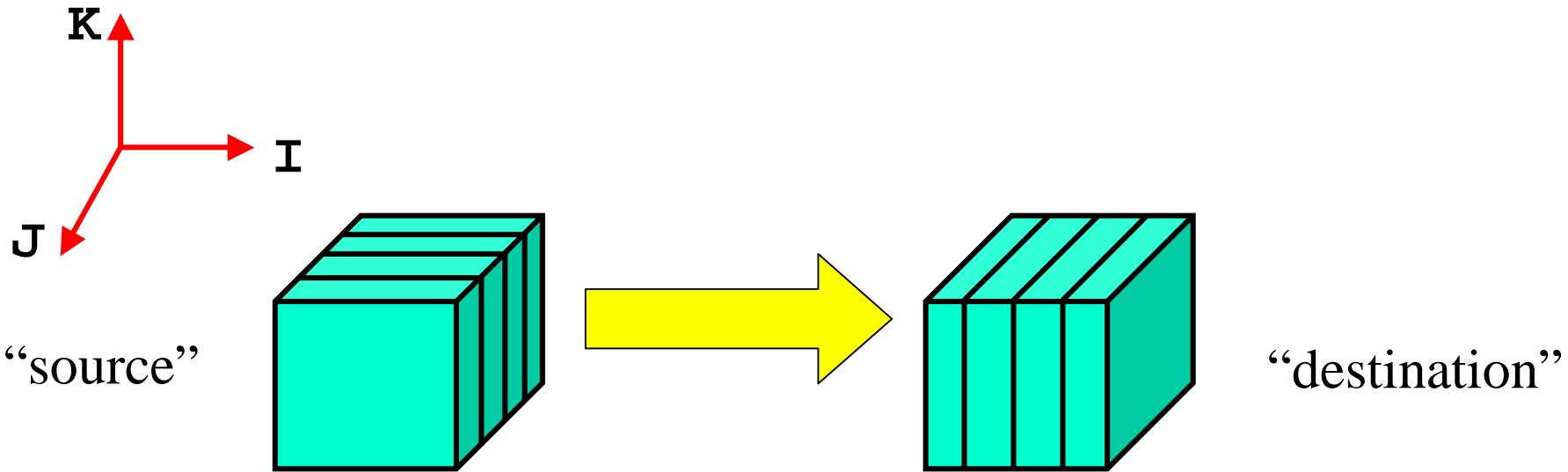
A Simple Hand-coded Exchange

```
C-----
C      STORE FROM WEST
      IF(MY_NEBS(4).GE.0)THEN
          IBEG=MYIS-IHALO
          IEND=MYIS-1
          CALL MPI_WAIT(IHANDLE(1),ISTAT,IERR)
          DO K=1,LL
          DO J=MYJS-JHALO,MYJE+JHALO
          DO I=IBEG,IEND
              ARR(I,J,K)=BUF0(I,J,K)
          ENDDO
          ENDDO
          ENDDO
          DEALLOCATE(BUF0,STAT=IER)
      ENDIF
C-----
C      STORE FROM EAST
      IF(MY_NEBS(2).GE.0)THEN
          IBEG=MYIE+1
          IEND=MYIE+IHALO
          CALL MPI_WAIT(IHANDLE(2),ISTAT,IERR)
          DO K=1,LL
          DO J=MYJS-JHALO,MYJE+JHALO
          DO I=IBEG,IEND
              ARR(I,J,K)=BUF1(I,J,K)
          ENDDO
          ENDDO
          ENDDO
          DEALLOCATE(BUF1,STAT=IER)
      ENDIF
```

A Simple Hand-coded Exchange

```
IF(MY_NEB(4).GE.0)THEN
    CALL MPI_WAIT(IHANDLE(4),ISTAT,IERR)
    DEALLOCATE(BUF3,STAT=IER)
ENDIF
C
IF(MY_NEB(2).GE.0)THEN
    CALL MPI_WAIT(IHANDLE(3),ISTAT,IERR)
    DEALLOCATE(BUF2,STAT=IER)
ENDIF
C
RETURN
END
```

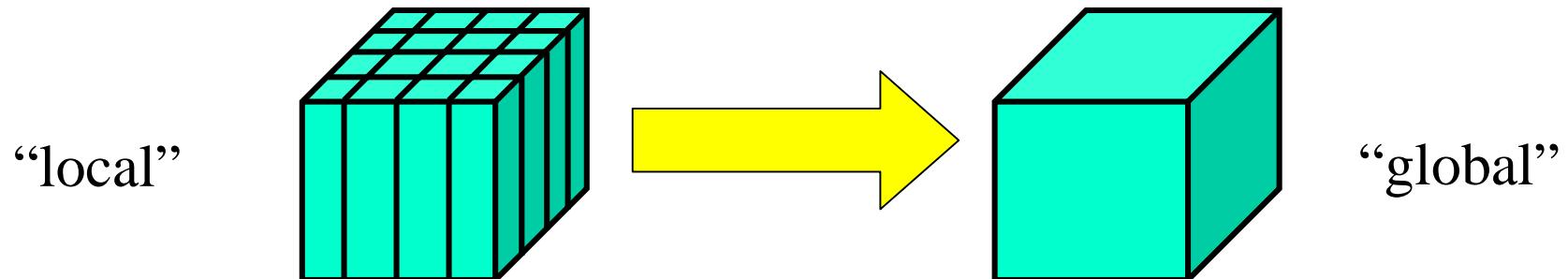
“Transfer”



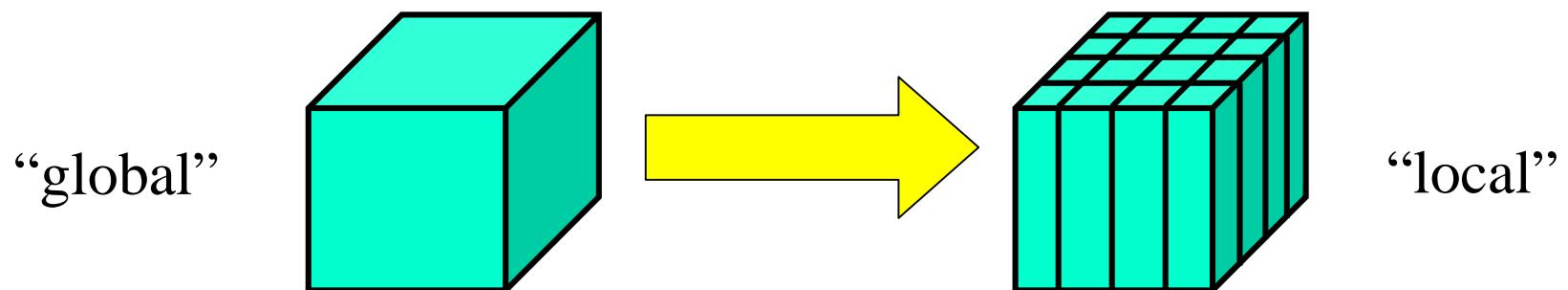
SMS Directive:

CSMS\$TRANSFER

Partial Parallelization



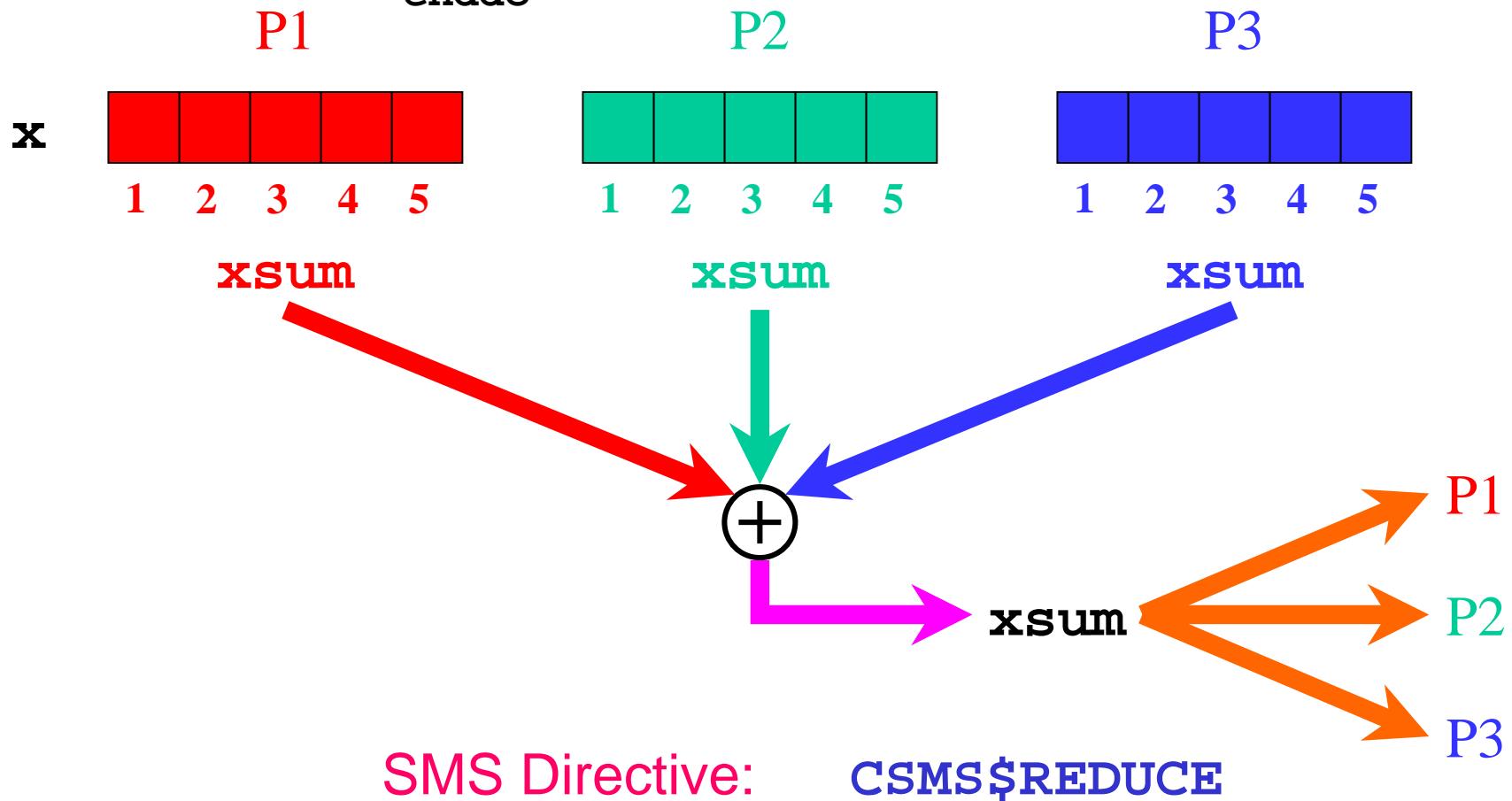
`CALL NOT_PARALLEL(...)`



SMS Directive: `CSMS$SERIAL`

“Reduction”

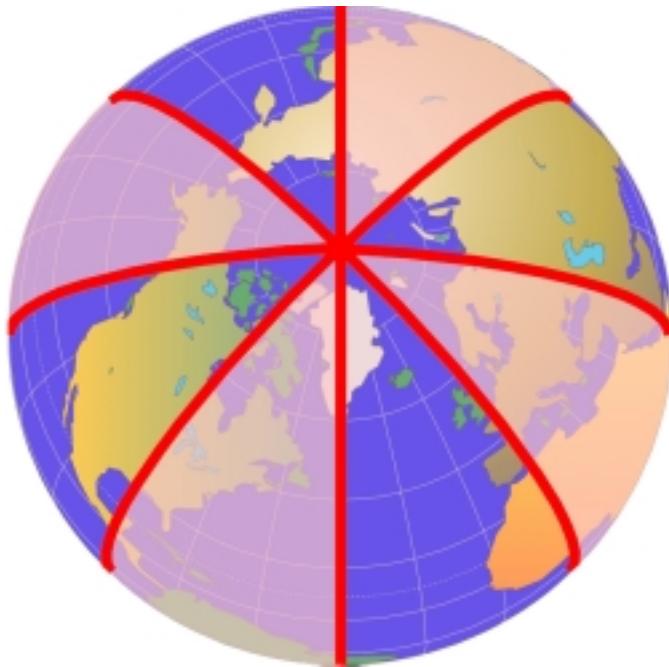
```
xsum = 0.0  
do i=1,15  
    xsum = xsum + x(i)  
enddo
```



What Does SMS Do?

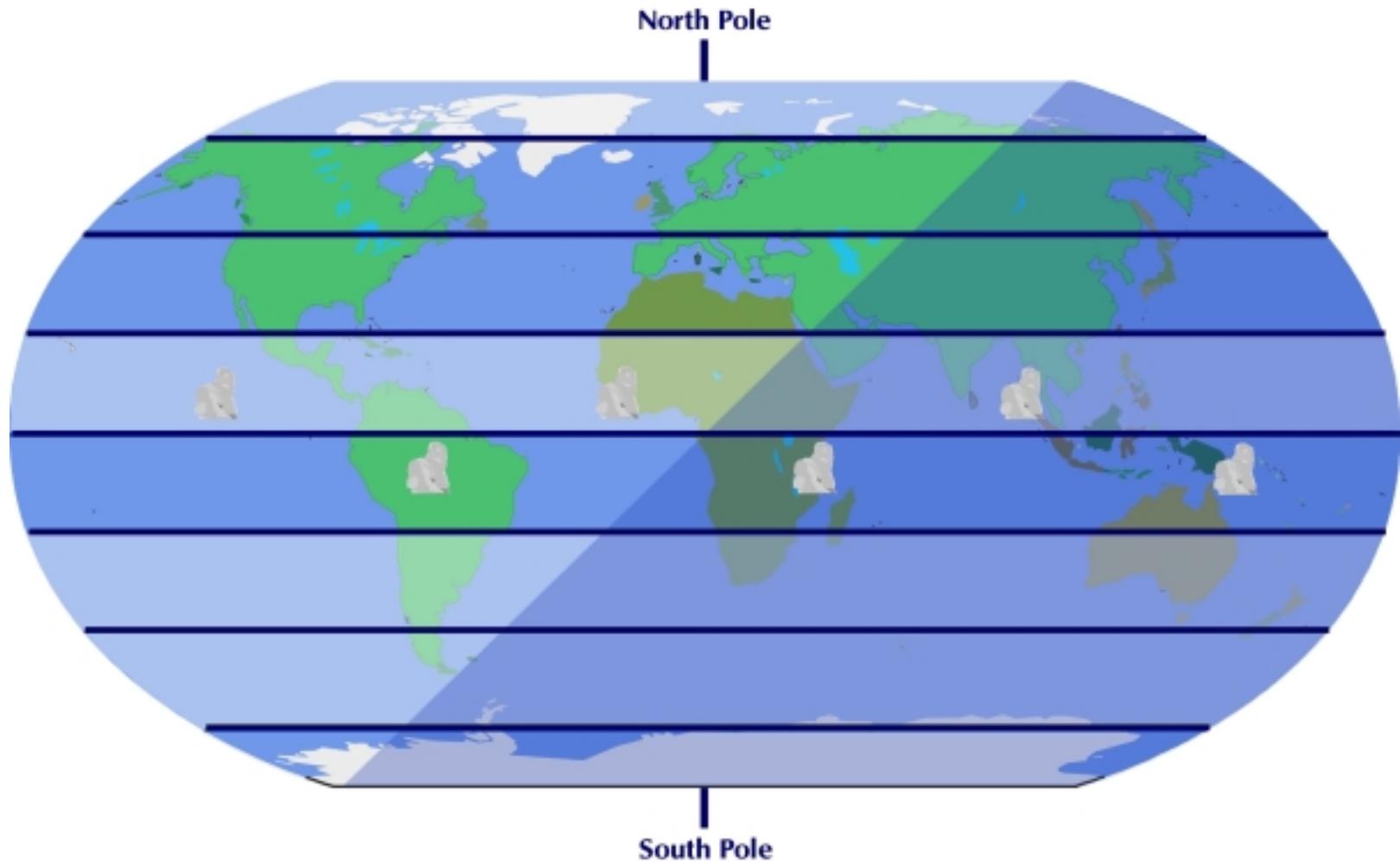
- Data decomposition
- Memory layout
- I/O
- Loop transformation
- Index translation
- Inter-process communication
- **Static load balancing**
- Performance optimization
- Debugging support

Static Load Balancing: Longitude Scrambling



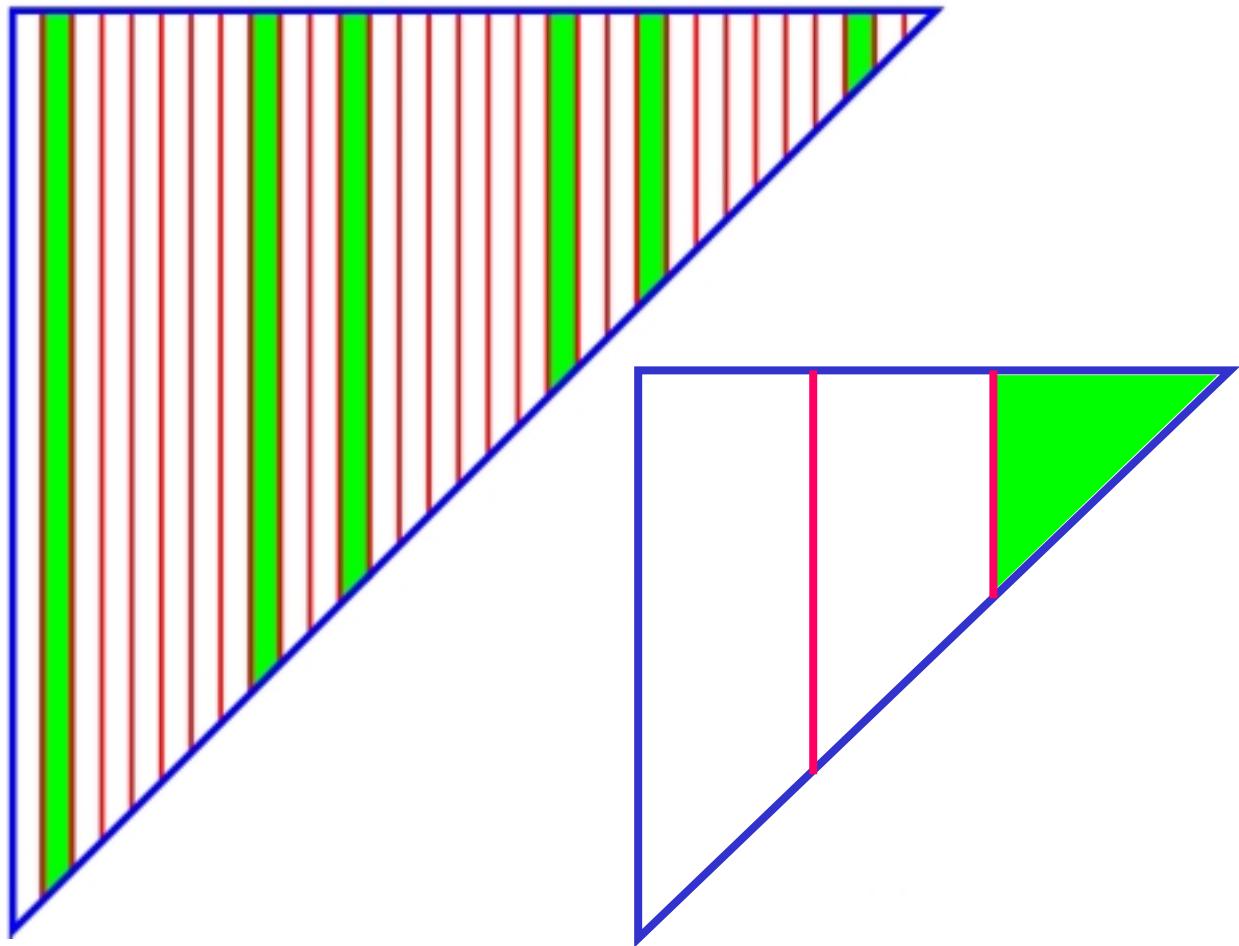
SMS Directive: keyword to **CSMS\$CREATE_DECOMP**

Static Load Balancing: Latitude Scrambling



SMS Directive: keyword to **CSMS\$CREATE_DECOMP**

Spectral Static Load Balancing (Scramble Fourier Wave Numbers)



SMS Directive: keyword to **CSMS\$CREATE_DECOMP**

What Does SMS Do?

- Data decomposition
- Memory layout
- I/O
- Loop transformation
- Index translation
- Inter-process communication
- Static load balancing
- **Performance optimization**
- Debugging support

Communication Aggregation

- Reduces latency for REDUCE, EXCHANGE, and TRANSFER
 - Easy to do
CSMS\$EXCHANGE(x,y,z)
 - SMS automatically combines messages

Trade Communication for Extra Computation

- Computation in the “halo” regions
 - Reduces the number of exchanges
 - Can reduce latency and bandwidth
 - Halo computation is “redundant”
 - Another process is already doing it
 - Performance tradeoff

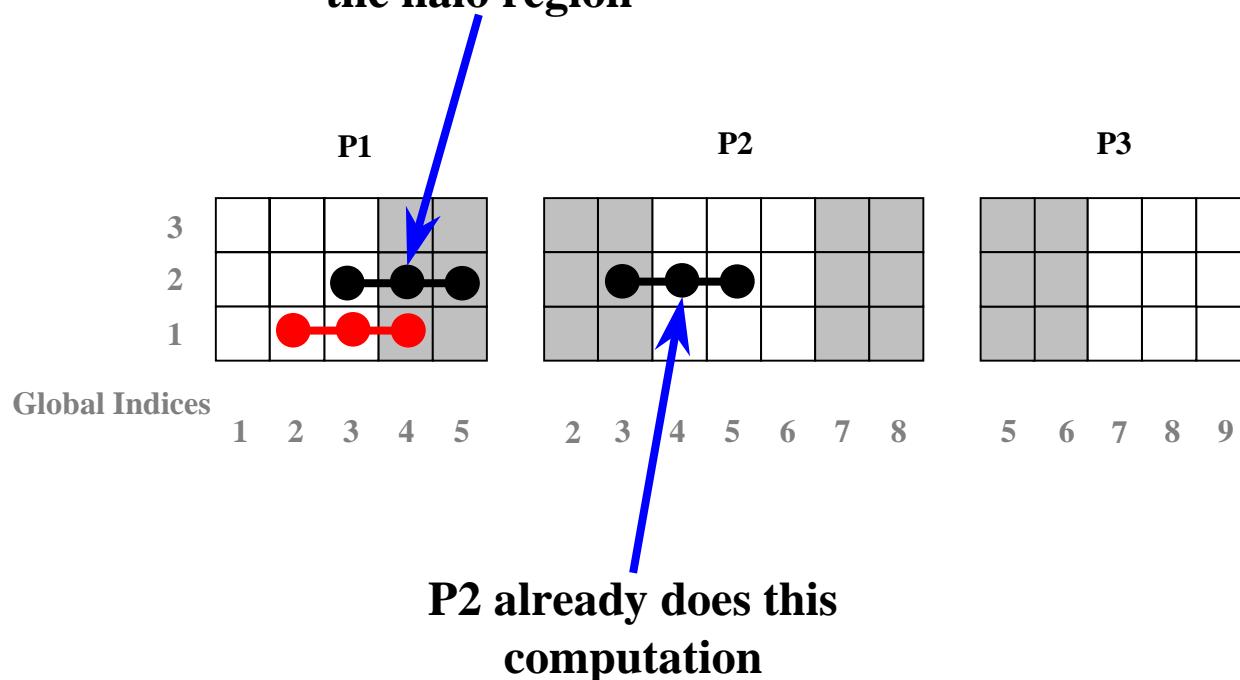
SMS Directive: `CSMS$HALO_COMP`

Computation in the Halo Regions

$$1) \quad y(i) = x(i-1) + 2.0 * x(i) - x(i+1)$$

$$2) \quad z(i) = y(i-1) + 2.0 * y(i) - y(i+1)$$

Stencil centered in
the halo region



I/O Performance Optimizations

- I/O server
 - Dedicate one process (CPU) to speed up I/O
 - Perform I/O in large blocks
 - Overlap writes with useful computation
 - Default behavior
- I/O without server
 - Select at run time via environment variable
 - Saves one CPU
 - I/O will be slower
 - Good when few CPU's are available
- Application-specific I/O tuning
 - Via directives, environment variables, or command line options

Machine-Specific Performance Optimizations

- Multiple algorithms for inter-process communication
 - Fastest method for target machine automatically selected
 - TRANSFER
- Machine-specific low-level optimizations
 - shmem on SGI in critical areas
 - etc.

What Does SMS Do?

- Data decomposition
- Memory layout
- I/O
- Loop transformation
- Index translation
- Inter-process communication
- Static load balancing
- Performance optimization
- **Debugging support**

Bitwise-exact Reduction

- Summation is not associative for floating-point numbers
 - Standard MPI "reduce" produces different results for different numbers of processors
- SMS supports bitwise-exact reduction for summation of real variables
 - Very useful for finding parallelization errors
 - Switch to faster inexact sums at run time via environment variable

Run-time Halo Checks

- Errors in parallel FDA codes often first appear as differences in the halos
- SMS supports automatic run-time halo checks
 - Directive: `CSMS$CHECK_HALO`
 - Switch checking on/off at run time via environment variable

SMS Code Example

```
integer IM, i
parameter(IM = 15)
CSMS$DECLARE_DECOMP(my_dh, <(IM/3) + 4>)

CSMS$DISTRIBUTE(my_dh, <IM>) BEGIN
    real x(IM), y(IM), xsum
CSMS$DISTRIBUTE END

C Begin executable code
CSMS$CREATE_DECOMP (my_dh, <IM>, <2>

    open (10, file = 'x_in.dat', form='unformatted')
    read (10) x
    close (10)

CSMS$PARALLEL(my_dh, <i>) BEGIN
    do 100 i = 3, 13
        y(i) = x(i) - x(i-1) - x(i+1) - x(i-2) - x(i+2)
100    continue
CSMS$EXCHANGE(y)
    do 200 i = 3, 13
        x(i) = y(i) + y(i-1) + y(i+1) + y(i-2) + y(i+2)
200    continue
    xsum = 0.0
    do 300 i = 1, 15
        xsum = xsum + x(i)
300    continue
CSMS$REDUCE(xsum, SUM)
CSMS$PARALLEL END
    print *, 'xsum = ', xsum
```

Hybrid SMS Solutions

- SMS + MPI
 - Use when SMS does not support a needed feature
 - First use low-level SMS Subroutines to get:
 - communicator
 - process id
 - Then mix MPI code with SMS directives
- SMS + OpenMP
 - Build with SMS only, OpenMP only, both, or neither
 - May want both on clusters of SMP's

Three NWP Models Parallelized with SMS Directives

- RUC (Rapid Update Cycle)
 - Finite Difference Approximation
 - Forecast portion of the model parallelized
- GFS (CWB Global Forecast System)
 - Based on spectral transform method
- QNH (FSL Quasi Non-Hydrostatic model)
 - Regional forecasts
 - Based on finite difference approximation

Results and Preliminary Performance

- Replaced hand-coded SMS parallel code with SMS directives
- Ran parallel models on newly procured “Jet”
- Parallel codes all produce correct answer
- No performance tuning done yet
- I/O times subtracted out
- PPP parallel codes performed as well as hand-coded solutions

RUC: PPP Modifications

- 12121 lines of code in forecast model
(excluding comments)
- 220 CSMS\$ directives added
 - BEGIN/ENDS counted as one directive
 - New AUTOGEN tool could automatically generate up to 1/3 of the remaining

RUC Performance

- 1-hour forecast
- 151x113x40, 40 km Resolution

Processors	Time (sec.)	Speedup	Efficiency
1	292	1.00	1.00
2	159	1.84	0.92
4	81.6	3.58	0.90
8	43.9	6.65	0.83
16	22.4	13.04	0.82
32	13.0	22.46	0.70
64	7.43	39.30	0.61

GFS: PPP Modifications

- 10224 lines of code in "computational core"
(excluding comments)
- 199 CSMS\$ directives added

GFS Performance

- T120, 18 levels
- Run for 12 hours

Processors	Time (sec.)	Speedup	Efficiency
1	2097	1.00	1.00
2	1070	1.96	0.98
4	552.9	3.79	0.95
8	293.0	7.16	0.90
16	160.2	13.09	0.82
32	91.40	22.94	0.72
64	55.97	37.47	0.59

QNH: PPP Modifications

- 6500 lines of code (excluding comments)
- About 150 CSMS\$ directives added

QNH Performance

- Run for 500 time steps
- 74x74x33, 20km

Processors	Time (sec.)	Speedup	Efficiency
1	414	1.00	1.00
2	196	2.11	1.06
4	92.5	4.48	1.12
8	49.0	8.45	1.06
16	24.6	16.83	1.05
32	14.1	29.36	0.92
64	8.80	47.05	0.74

“Hand-coded” SMS QNH CONUS

- Run for 500 time steps
- 258x194x33, 20 km

Processors	Time (sec.)	Speedup	Efficiency
10	233	10.00 *	1.00 *
20	111	20.99	1.05
30	74.1	31.44	1.05
56	39.1	59.59	1.06
80	27.9	83.51	1.04
100	22.8	102.19	1.02
200	13.8	168.84	0.84

SMS Summary

- SMS directives are much simpler than hand-coded parallelization using MPI
- Single source for serial and parallel code
- SMS works on both shared and distributed memory machines
- Performance optimizations are easy to use

SMS Summary (contd)

- Performance of translated code is comparable to low-level SMS subroutines (PPP adds negligible overhead)
- Currently working to directly compare hand-coded MPI to SMS directive solution
 - MPI codes do not always use optimizations provided by SMS
 - If necessary, SMS can be mixed with MPI and/or OpenMP to further fine-tune performance

SMS Summary (contd)

- SMS directives support incremental parallelization
- Unformatted I/O requires no directives
- Serial data ordering is preserved in files
- Native or Portable File Formats Available
- Extensive documentation is now available

www-ad.fsl.noaa.gov/ac/sms.html

Ongoing Efforts

- 2000
 - Full directive-based support for nested, coupled, and multi-grid models
 - Support for general linear interpolation
 - Typhoon model for Taiwan CWB
 - Support for Fortran90 syntax
 - Continued performance optimization
 - Performance testing in conjunction with OpenMP
 - Simple load balancing for FDA models
 - More debugging support
 - Establish other collaborations

Future SMS Development Plans

- 2001 – ?
 - Dynamic load balancing
 - PPP-generated mix of SMS and OpenMP
 - May be a good idea on some SMP cluster architectures
 - Continued performance optimization
 - Computer aided dependence analysis